ISSN 2007-9737

Sentence Generation Using Selective Text Prediction

Samarth Navali, Jyothirmayi Kolachalam, Vanraj Vala

Samsung R&D Insitute, Bengaluru,
India

{s.pnavali, jyothi.kolac, vanraj.vala}@samsung.com

Abstract. Text generation based on comprehensive datasets has been a well-known problem from several years. The biggest challenge is in creating a readable and coherent personalized text for specific user. Deep learning models have had huge success in the different text generation tasks such as script creation, translation, caption generation etc. Most of the existing methods require large amounts of data to perform simple sentence generation that may be used to greet the user or to give a unique reply. This research presents a novel and efficient method to generate sentences using a combination of Context Free Grammars and Hidden Markov Models. We have evaluated using two different methods, the first one is using a score similar to the BLEU score. The proposed implementation achieved 83% precision on the tweets dataset. The second method of evaluation being a subjective evaluation for the generated messages which is observed to be better than other methods.

Keywords. Text generation, sentence generation, context free grammar, CFG, hidden Markov model, HMM, selective text prediction.

1 Introduction

Machine Learning has found a lot of application in almost all the major industries such as finance, IT, healthcare, etc. Machine Learning has started to replace the traditional methods mainly because of the power of these algorithms. They are used for a variety of tasks such as generation, classification, localisation etc. They learn features from the training data and then use what it has learnt to perform the respective task. They were not preferred initially as there wasn't enough computational power to run these algorithms. With computational power easily available these days, machine learning has emerged as the go-to solution.

Natural Language Processing(NLP) is one such application. In this application machine learning models try to gain some understanding from textual,voice data and then use it for various other applications. Machine learning for NLP may be used to generate text, identify parts of speech, to perform Named Entity Recognition(NER) etc. These kinds of tasks are used to gain intelligence from the text, voice based data and then utilise this information as and when required.

One of the tasks that come under NLP is generation of text. This generation is based on some existing textual data. At times it so happens that somebody wants to caption an image, but that person doesn't know what would be a good caption or when somebody wants to write a poem which seems as if it's written by Robert Frost. All these are tasks that involve generation of text. With the help of Machine Learning one can use the previous occurrences of the task and learn from it and generate text.

Neural Networks have been found to attain state-of-the-art results in a majority of the natural language processing tasks such as sentiment analysis, text generation etc. Within NLP, quite a number of tasks involve generating text based on some input data.

Neural Networks are really powerful and can gain more insights from the input data as compared to the other machine learning algorithms.

In the recent years Recurrent Neural Networks(RNN) and Long Short Term Memory Networks (LSTM) have taken over in such tasks. They have the power to remember the contexts from before and use that context details in generating text. Text is generally generated from these models that takes a sample from a distribution that is conditioned on the previous words and the hidden state consists of some representation of all the words generated so far. At times they are trained with a method called as teacher forcing, where the ground-truth words are fed back into the model for the generation of text. This causes problems as the model is forced to condition on sequences that were not initially observed during the training time. This leads to unpredictable outcomes in the hidden state of the RNN. Also as they require large amounts of data to train and generate presentable sentences, they were not preferred because of the availability of less data.

As the target for this research work is to generate simple sentences, it's not worth going through all the disadvantages of the RNN. Also as for such tasks we are not utilising the true power of RNN's and LSTM's, so using them does not make sense. So for these cases this research work describes a method where we combine Context Free Grammar's and Hidden Markov Models to generate sentences.

A **Context Free Grammar (CFG)** is a set of recursive rewriting rules which are used to generate a variety of strings. It is a quadruple (N,T,P,S) where:

— N is a set of non-terminal symbols.

— T is a set of terminals where N intersection T = NULL.

— P is a set of rules, P: N -> (N U T), i.e., the left-hand side of the production rule P does not have any right context or left context.

— S is the start symbol.

We first start off with the starting symbol and then replace each of the starting symbol with the rule associated with that symbol.

Then proceed until no more substitutions can be made or until the desired terminal state is reached. CFG's are used to generate patterns of strings, pattern matching etc.

Hidden Markov Model (HMM) is a method for representing probability distributions over a sequences of observations. It gets its name from two main properties. The first one being, the prediction for the current state comes from a state St that is hidden from the user. The second one being the assumption that the state of the hidden process satisfies the Markov property that is the current state depends solely on the previous state and none of the states before that.

2 Related Work

We have many deep learning models detecting the text based on available corpus by providing input text and grammar corrections. However, these networks have fallen out of favour for modelling sequential text data, as they require context lengths, more computation, more data and previous hidden state summary of different time stamps. In the neural networks [4] approach which chooses different entities to predict next words, neural networks has to be trained and data is generated based on previous entity provided which results in additional memory being consumed. This method performs something similar to teacher forcing models. These models are trained by feeding the ground truth words to the network for generating the different parts of the sentence.

To avoid this we have popular variations of Recurrent Neural network models such as long short-term memory (LSTM) and Gated recurrent unit (GRU) where text generation happens based on usage of words and its probable occurrence from generated sentence. In such models, all possible words are predicted and appended for forming sentences, reassessment will be performed to results later. Though possible sentence generation is high, evaluating all the possible sentences takes more computational time and memory.

In models where Generative Adversarial Networks (GANs) are used, the generator is trained to produce high quality samples but accuracy obtained is more for images than for text sequences.

Some of other deep learning models where RNN's are designed to process sequential information with help of previous state i.e. memory. At every processing step, input sequences, accumulating information from past are presented to RNN, which modify network state. LSTM [2] sequence-to-sequence models are a special class of RNN's known for their ability to effectively learn long-term dependencies in sequences. These models maintain a forget gate, which determines how much of previous cell state should be passed on current time stamp. Sequence to sequence models are applied in video captioning, speech recognition etc. These models use encoder-utilizing words of source sentence in forming context vector, which summarizes semantics of a sentence and decoder for operating on this semantics vector to generate required translation words. These models give less execution efficiency and are compute intense.

As deep neural network models are very dense in computing [1], we have popular methods of analysis. Hidden Markov Model (HMM) [3] which estimates the probability of text occurrence in given position, based on sequence of preceding values. Number of occurrences of words length (k+1) in learning vector is calculated. Transition matrix approach is used for getting probable occurrences of data under different conditions. For smaller values of k this approach is efficient but, as k grows transition matrix also grows and this will cause the problem of insufficient memory to store the vectors.

Context Free grammar (CFG) [5] tool kits are also available which generated based on usefulness and reachability of text. Programming languages laboratory at university of Caligari provided an online tool for context Free grammar checker to check basic properties of context free grammars, where in the tool generates not more than 20 sentences, which are the first ones ordered by sentence length. Generated sentences are too simple. Some of the best CFG tools are used in several research projects like SAQ and grammar testing methods where Purdom's algorithm and CDRC-P algorithms are used but still failed in generating appropriate text sentences. Our approach is quite simple, we combine CFG's and HMM models where context free grammars are generated from the input sentences and the structures are stored, which results in higher accuracy and meaningful sentences for the user.

3 Proposed Methodology

We propose the use of CFG's to understand the structure of the sentences from the input data and use HMM to predict the words based on the CFG. We use both of them together in tandem to perform a selective prediction of words. The prediction is a two-step process, we use the CFG's to identify the type of word that will be predicted next (such as ADJ, VERB, PROPN etc.) and then with the help of a second order markov chain we make a prediction of the next word that is of the type expected in the CFG.

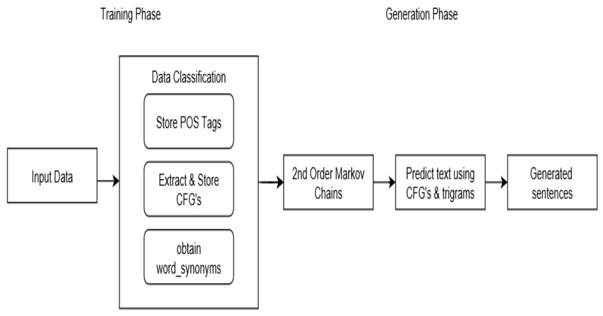

Fig. 1. Proposed methodology to generate sentences

For example if we have to generate sentences for the CFG [ADJ, NOUN, PUNCT]. As we are using a second order markov chain we choose the first two words randomly from the part of speech (POS) as mentioned in the CFG. In this case we choose words from the part of speech type of ADJ and NOUN. While selecting the first two words we should make sure that it is a part of a valid second order markov chain (will be explained later

Algorithm 1: Procedure for sentence generation

```
POS tagging on input data
POS[tag] = word_list
obtain markov_chains from input data
obtain CFG_sentence_structure from input data
for each CFG in CFG_sentence_structure
   final_sentence = ""
   w1 = random_word(POS[CFG[0]])
   w2 = random_word(POS[CFG[1]])
   while (w1,w2) not in keys(markov_chains) :
      w1 = random_word(POS[CFG[0]])
      w2 = random_word(POS[CFG[1]])
   for i from 2 to len(CFG)
      if markov_chains[(w1,w2)] intersection POS[CFG[i]]
                                                  exists
         next_word = random_word(POS[CFG[i]])
         while next_word not in markov_chains[(w1,w2)]
            next_word = random_word(POS[CFG[i]])
         final_sentence += " " + next_word
         w1 = w2
         w2 = next_word
   print final_sentence
```

Fig. 2. Algorithm that is used to generate sentences

in detail). So in this case, assuming we choose the words 'Good' and 'Morning', as it is part of a valid second order markov chain, we proceed. The next word predicted should be of the type PUNCT and it should be preceded by 'Good Morning'. We randomly choose the next word from the second order markov chains, assuming the selected word is '!'. As we have reached the end of the CFG the sentence generation is over and the final sentence output is 'Good Morning!'.

To provide more variety to the generated sentences, synonyms of the predicted word are placed in the generated string. Once the word is predicted, synonyms of that particular word are obtained and put into the generated sentence. As synonyms have the similar meaning to the original predicted word even if we do substitute the actual word with the synonym word the semantics of the generated message will not vary.

This research work presents two different phases, the **training phase** and the **generation phase**. The training phase involves the preparation of the data and the creation of all the required components for the generation phase. These include the POS tags, the CFG list etc.

3.1 Training Phase

In this phase we extract all the necessary details from the input data and then store them for further use in the generation phase. The first step in this phase is the POS tagging of the input data. This is important as it would help in understanding the structures of the sentence in the input data and also would get rid of ambiguity by realising all the parts that each word takes, for example the word 'sun' can either be a verb or a noun. Part of speech also helps us understand what the word means in that particular context.

So the first step is tag the input data to with the POS tags, this is done with the help of the spacy tool. Spacy internally tokenizes the text and gives the POS tags to each token. For the tweets input data, about 13 unique parts of speech have been identified and used for prediction. These POS tags are used for two different tasks. Firstly they are used to store all the words for the 13 unique identified parts of speech and secondly to create the CFG's for all the sentences in the input data.

The first use of the POS tagging is to store all the words and their respective tags. A dictionary is created where the keys are the 13 unique parts of speech and the values are all the words having the respective part of speech. This dictionary will be used in the selective prediction of words, that is the next selected word is not completely random and that it will be guided by the CFG's and this particular dictionary.

The second use of the POS tagging is to obtain all the sentence structures from the input data. As we take the assumption that the input data has the right sentence structure, we will retain the same sentence structure and use it to generate sentences. Each sentence from the input data comprises of one right sentence structure and all of them are stored. Each CFG/sentence structure will be used to generate sentences and they are one of the main requirements as the predictions are based on these structures. Only unique sentence structures are used for the prediction. These two steps consist of the training phase of this work.

3.2 Generation Phase

This is the phase where the actual word prediction happens. The input data is used to store the second order markov chains. Second Order Chains are used used mainly because they provide the right amount of variations in the generated sentences and also makes sure that the generated sentences are not truly random. Choosing a smaller chain causes the generated sentence to be truly random and choosing a longer chain will reduce the variety by constricting the generation to a specific set of words.

These chains help us in choosing on what the sentence begins with, also as the input data already contains the right lemma of the word, we will reuse this when we predict words. So we will have the right lemma in the right context when we predict words. Having the right lemma of words makes sure that the sentence formation and more importantly the sentence sounds right. At this point we have the list of CFG's, POS dictionaries, and the second order markov chains, that is all of the key items required for prediction.

For each CFG there exists a start state. This start state marks how the sentence begins. In our case the start state is nothing but a part of speech. So we randomly pick a word (w1) from the POS dictionaries for the start state and then build from it. As we are using second order markov chains we repeat the same step that is we randomly obtain another word (w2) as per the second part of speech in the CFG. If the pair [w1, w2] doesn't have any word following it which is the part of speech of the next position in the CFG, then we reselect w1 and w2 and move forward. If it does have one or more words following [w1, w2] that have the part of speech as expected in the CFG, we randomly select until we obtain a word (w3) that is of the expected part of speech and then append that word to the generated string.

As we are using a second order markov chain the searching key changes from [w1, w2] to [w2, w3] and then the process continues. In case there are no words following the pair [w2,w3] that is of the expected part of speech then we just ignore that part of speech and move to the next expected part of speech. Once we predict a word, to provide more variety in the strings that are generated we have synonyms for the words. We once again randomly pick a synonym each time and that particular synonym is appended to the final generated string. As the synonyms have the similar lemma and have the similar meaning, replacing them with the actual predicted word will not a cause a huge change in the semantics of the generated sentence or the structure of the sentence.

3.3 Dataset

The dataset that was used was the Good morning Tweets Dataset. This dataset consists of all the tweets that contain the phrase ‘Good Morning’ . The dataset consists of about 3000 tweets. The tweets were formatted by removing the retweets and the duplicates. The data was further formatted by getting rid of the url links. All the hashtags were removed as they are not uselful in this particular task. The dataset is available online[6]

4 Evaluation Criteria

We use two different evaluation criteria. The first one is a score very similar to that of the Bilingual Evaluation Understudy (BLEU) score. In this scoring method, from the generated text we calculated the number of second order markov chains that are present from the initial input data. All second order markov chains are obtained from the generated dataset and counted if that chain is present in the input data. If it is present then we know that the predicted text has the right lemmas for the word and that the context will mostly be maintained, so we count it as a right chain, else it is identified as a bad chain.

One side effect we noticed from this was that, in the predicted text we had synonyms instead of the actual predicted words. As some of the synonyms were not present in the initial vocabulary of words, it was giving a false score in some of the cases but it's not a wrong thing because that sequence has similar meaning but is being wrongly penalised. So to overcome this effect we replace all the key words with the actual word that had been predicted.

As one word can be part of multiple synonyms, there will be a clash as to through which predicted word we got the particular synonym. To solve this problem we store each synonym with a particular version (such as 1.0 or 1.1) to depict which predicted word the synonym had come. So based on the version of the synonym we replace it with the respective predicted word. And then go forward with this criteria:

$$Precision = \frac{n_v}{n_a}, \tag{1}$$

where, n_v = number of valid chains in generated sentences, n_a = number of chains in generated sentences.

As seen from above, the score is the number of matching second order markov chains divided by the total number of markov chains.

The second criteria is a subjective evaluation of the generated sentences. A 1000 sentences were randomly selected from the generated sentences. Each sentence was evaluated on 3 main key points:

— Sentence structure,

— Vocabulary in the sentence,

— Semantic meaning preservation.

Sentence Structure means to check if the general structure of the sentence is maintained or not. We check how the generated sentence is arranged grammatically, that is evaluate if the parts of speech are placed in the right part of the generated sentence.

Vocabulary is the second stage of checking. In this we check the usage of the words. Also the check of the right lemma in the different contexts verifies that the vocabulary in the generated sentence is right.

The third and final check in this evaluation criteria is the revival of the semantic meaning from the input data. This is to check if the semantic meaning of the generated sentence is similar to the one form the input sentence. Here the generated sentence is compared with the ground truth and if they do have similar semantics then it is considered as a good generated sentence.

All the above three criteria are based on comparison with the ground truth. If sentence satisfies all the above criteria then it was counted as a valid sentence.

5 Results

The evaluations were done on five different models. The five models are as given below:

— The first model generated sentences using only a Context Free Grammar.

— The second model generated using a Hidden Markov Model.

— The third one was another implementation which uses the above two technologies.

— The fourth one was our implementation which combines both CFG's and HMM's.

— The fifth one being a LSTM Model.

As stated above we use two methods for evaluation. The first one is a score similar to a BLEU score. The second method is a subjective evaluation.

In the first method we create all second order markov chains from the generated sentences and observe how many of these chains have been observed before. With this method we have observed a precision of 83%.

Table 1. These results depict the amount of second order markov chains that have been retained from the ground truth. Results are shown for all five models where the fourth model is our implementation

Dataset	Technique	Chains Retained
Good Morning	CFG	25.9 %
Tweets	HMM	**86.2 %**
	CFG & HMM	30.5 %
	CFG & HMM II	**83.7 %**
	LSTM	28.8 %

As seen from Table 1 a comparison was made between the number of second order markov

ISSN 2007-9737

chains that were retained from the original dataset. The Hidden Markov Model performs the best as it works solely on the second order markov chains and then comes our implementation. It performs much better than the alternative implementation that also uses both HMM and CFG to predict the text.

Table 2. Subjective evaluation was performed on the five different models. The fourth model is our implementation

Dataset	Technique	Score
Good Morning	CFG	32.3 %
Tweets	HMM	35.5 %
	CFG & HMM	40.7 %
	CFG & HMM II	**52.2 %**
	LSTM	49.8 %

The second method we used was a subjective evaluation. As seen from the table 2 , our implementation performs much better than the other methods. This is a result of combining two methods that perform fairly poorly and to produce a model that performs much better. When we combine the two methods we are utilising the advantages of each model in our own model. The biggest advantage is that this model will work with less input data.

6 Conclusion

Neural Network models perform really well, but are compute intensive and require a large amount of data. If there is no large amount of data then they do not perform well. If there is no computation power then these class of algorithms go for a toss. The problem of less data for training can be handled using CFG and HMM to predict text. But individually they do not perform well, but when combined together they perform much better. The next good part is that not a lot of computation power is required in the proposed method. It was observed when combined together, the proposed method was able to retain the semantic meaning, the grammatical structure and sentence structure from the original sentences.

The proposed method has a low memory footprint as not all possible sentences are generated and then evaluated. In this case we just pick one of suitable words based on the context and the structure. In the earlier methods all possible words are appended to the string rather than word, in this way the proposed method has a low footprint. In this case the method does not have to go through the problems of remembering data from the past as it is involved with generating simple sentences not based on any data from the past.

The proposed work can be used in the cases of greeting the user in a unique way each time, or in the case of giving an automated reply, or when the system wants to remind the user to do something. When the variety is given to the user then it makes the user feel good about the system that they have. This work has shown that the proposed method can achieve significant results and can be used in the above mentioned methods.

Going forward, the proposed novel method can be combined with deep learning techniques so that we can attain the accuracy that is achieved (by Neural Networks) with lesser data and as well as a lower memory footprint. We can take the positive things from both the models and work on coupling them together and obtaining a model that can achieve very high accuracies.

References

1. **Fedus, W., Goodfellow, I., & Dai, A. M. (2018).** Maskgan: better text generation via filling in the_. *arXiv preprint arXiv:1801.07736*.
2. **Sen, S. & Raghunathan, A. (2018).** Approximate computing for long short term memory (lstm) neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 37, No. 11, pp. 2266–2276.
3. **Szymanski, G. & Ciota, Z. (2002).** Hidden markov models suitable for text generation. *WSEAS International Conference on Signal, Speech and Image Processing (WSEAS ICOSSIP 2002)*, pp. 3081–3084.
4. **Xie, Z. (2017).** Neural text generation: A practical guide. *CoRR*, Vol. abs/1711.09534.

5. **Xu, Z., Zheng, L., & Chen, H. (2010).** A toolkit for generating sentences from context-free grammars. *2010 8th IEEE International Conference on Software Engineering and Formal Methods*, IEEE, pp. 118–122.

Article received on 18/01/2019; accepted on 04/03/2019. Corresponding author is Samarth Navali.